Nat. Volatiles & Essent. Oils, 2021; 8(4): 1207-1231

NVEO
Natural Volatiles &
Essential Oils

Efficient file downloading and distribution using grid computing for intelligent applications

M Niranjanamurthy[1], Dayananda P[2], Amulya M P[3], CH Ram Mohan Reddy[4]

[1]Assistant Professor, Department of Master of Computer Applications, M S Ramaiah Institute of Technology(Affiliated to VTU, Belgaum), Bangalore, India. E-Mail: niruhsd@gmail.com

[2]Professor, Department of Information Science and Engineering, JSS Academy of Technical Education, Bangalore-560060, India, Email: dayanandap@gmail.com

3Assistant Professor, Department of Computer Science and Engineering, BGS Institute of Technology/Adichunchanagiri University, B G Nagar- 571448, Mandya, India Email: ammu.raashi@gmail.com

[4]Associate Professor, Department of Computer Applications, BMS College of Engineering, Bengaluru, Karnataka 560019 India E-Mail: prof.crmr@gmail.com

Abstract: Grid computing works very well for sorting or organizing large amounts of data that is incorrectly grouped or needs to be spaced. It operates on the concept of several hundred interconnected and networked computers that function like stacked software and manage work like a supercomputer. An intelligent system is a machine with a built-in computer connected to the Internet that has the ability to collect and analyze data and communicate with other systems. To create a framework for distributed query engine which supports global grid computing using dynamic task scheduling by centralized resource management and automatic task allocation and result generation in most effective, less time, with data security. Research work is to recognize the implications for software quality of various decisions which must be made in the process of specifying a distributed system. The core logic of the work is to build a generic distributed system which supplies a friendly user API allows remote execution. In this paper represent the efficient file downloading and distribution using grid computing for intelligent applications.

Keywords: Grid computing, Intelligent application, System Architecture, Algorithm, Implementation, System Manager

1. Introduction

A traditional operating system on a standalone computer controls the hardware of that computer, and provides a nice abstracted interface to applications that run on that computer. A network operating system works with a standalone operating system to provide communication facilities to the applications that run on that computer. A network operating system usually is not defined separately but combined into the overall lump of things called the operating system. An application running on a

computer with network connectivity has to know what other computers are out there and how to communicate with them. A distributed operating system takes the abstraction to a higher level, and allows hides from the application where things are. The application can use things on any of many computers just as if it were one big computer. A distributed operating system will also provide for some sort of security across these multiple computers, as well as control the network communication paths between them. A distributed operating system can be created by merging these functions into the traditional operating system, or as another abstraction layer on top of the traditional operating system and network operating system. Any operating system, including distributed operating systems, provides a number of services. First, they control what application gets to use the CPU and handle switching control between multiple applications. They also manage use of RAM and disk storage. Controlling who has access to which resources of the computer (or computers) is another issue that the operating system handles. In the case of distributed systems, all of these items need to be coordinated for multiple machines.

As systems grow larger handling them can be complicated by the fact that not one person controls all of the machines so the security policies on one machine may not be the same as on another. Some problems can be broken down into very tiny pieces of work that can be done in parallel. Other problems are such that you need the results of step one to do step two and the results of step two to do step three and so on. These problems cannot be broken down into as small of work units. Those things that can be broken down into very small chunks of work are called fine-grained and those that require larger chunks are called coarse-grain. When distributing the work to be done on many CPUs there is a balancing act to be followed. No one want the chunk of work to be done be so small that it takes too long to send the work to another CPU because then it is quicker to just have a single CPU do the work, You also don't want the chunk of work to be done to be too big of a chunk because then you can't spread it out over enough machines to make the thing run quickly.

We look at a number of distributed systems that have attempted over the distinction between local and remote objects, and show that such systems fail to support basic requirements of robustness and reliability. These failures have been masked in the past by the small size of the distributed systems that have been built. In the enterprise-wide distributed systems foreseen in the near future, however, such a masking will be impossible. If not planned properly, a distributed system can decrease the overall reliability of computations if the unavailability of a node can cause disruption of the other nodes.

Troubleshooting and diagnosing problems in a distributed system can also become more difficult, because the analysis may require connecting to remote clients or inspecting communication between clients. Many types of computation are not well suited for distributed environments, typically owing to the amount of network communication or synchronization that would be required between nodes. If bandwidth, latency, or communication requirements are too significant, then the benefits of distributed computing may be negated and the performance may be worse than a non-distributed environment.

A paradigm for the system and software design of distributed systems is presented with application to an actual large-scale computer network. In this synopsis, a design principle is offered with particular reference to how they can be applied to the design of distributed systems. The aim is focused to an implementation of how to make design decisions about distributed systems in a way which will enhance maintainability and understandability of the software and, at the same time, result in good system performance. The aim is to recognize the implications for software quality of various decisions which must be made in the process of specifying a distributed system. The core logic of the project is to build a generic distributed system which supplies a friendly user API allows remote execution. The prototype is intended for clients who need to execute tasks over more than one computer without getting involved with complex networking APIs. Using the system will allow the client to run his tasks produced over one computer and the system will take care of distributing the tasks over all computers connected to it by their priority in order to be executed in parallel and return the tasks' results to the same computer the developer use deploying Round Robin Algorithm. The system is platform independent; means it can run over every platform since it is written in Java programming language. Moreover, the system can be used in environments where some machines are behind firewalls.

Distributed computing refers to the means by which a single computer program runs in more than one computer at the same time. In particular, the different elements and objects of a program are being run or processed using different computer processors.

Distributed computing is similar to parallel computing and grid computing. Parallel computing, though, refers to running a single program using a minimum of two processors that belong to one computer. Grid computing, on the other hand, refers to a more dedicated distributed computing setup - one whose computer 'members' are especially dedicated to the program being processed.

The terms "concurrent computing", "parallel computing", and "distributed computing" have a lot of overlap, and no clear distinction exists between them. The same system may be characterized both as "parallel" and "distributed"; the processors in a typical distributed system run concurrently in parallel. Parallel computing may be seen as a particular tightly-coupled form of distributed computing, and distributed computing may be seen as a loosely-coupled form of parallel computing. Nevertheless, it is possible to roughly classify concurrent systems as "parallel" or "distributed" using the following criteria:

- In parallel computing, all processors have access to a shared memory. Shared memory can be used to exchange information between processors.
- In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.

2. Literature review

With the quick improvement of new innovation in different fields, the interest for power is expanding step by step. The quantity of gadgets associated with the advanced lattice and the measure of information created have expanded dramatically. The conventional cloud-based concentrated enormous information preparing mode can't completely address the issues. Edge processing, another figuring model that moves part or the entirety of the first distributed computing undertakings to the area of the information source, has slowly drawn in broad consideration from different industries.[1]

shrewd framework innovations remember complex apparatuses for request to screen and control the force framework in both manners from power stations to end-clients or bad habit versa.So that numerous weaknesses and force breakdown can be identified ahead of time and important alerts can be taken. Also, the keen matrix framework offers the checking and the administration of the electrical energy from age to end-client, and gives shrewd metering, vehicle to lattice association just as joining of the environmentally friendly power to the network. Additionally, the proficient utilization of force sources with least misfortune and least illicit use is likewise taken care of in shrewd network technology.[2]

The intense expansion in the product PC and organization execution for the last age has a resultant of quicker equipment and more complex programming. However, the supercomputers of the current age are as yet unequipped for tackling the current issues in the field of science, designing, and business. This issues emerges as a solitary machine can't work with the accessibility of different heterogeneous assets needed to determine the emergency. Creator recommended Grid computing.[3]

Zeroed in on the issues that the advancement of lattice application is troublesome and the sort of found assets is confined when straightforwardly utilizing globus tool compartment API, a few innovations like matrix holder, network module, framework object pool, framework asset connector and the subgrid correspondence convention for registering framework are presented, the part techniques and hiberarchy of figuring network appropriated middleware.[4]

The Network processing, another registering framework, gives an expected answer for saddle inactive workstations and different assets in an adaptable way in grid computing. [5]

As another processing worldview, edge figuring has showed up in the public field of vision as of late. Attributable for its potential benefits of low deferral and quick reaction, edge processing has gotten a significant aide of distributed computing and has brought new freedoms for different savvy applications like the keen framework, the keen home, and the keen transportation.[6]

The conventional force lattice framework is advancing toward a keen matrix framework, which will further develop the client experience.[7]

The Smart Grid is another age power framework that incorporates with correspondence data innovation which includes a ton of development, trend setting innovation, and applications. This has immediately transformed into additional high level turn of events, as it is confronting difficulties like large information issues, energy robbery, and so on, effectively.[8]

A Smart Grid (SG) is an electrical framework similar as heritage power network with versatile and unavoidable two-way correspondences, convenient control abilities, enormous scope coordination of disseminated assets and productive utilization of assets. The SG gives the highlights of inescapable shrewd observing innovations, programmed gear flaw detecting and self-healing.[9]

Productive booking is a vital worry for the strong execution of execution driven Grid applications, like work processes. Many rundown heuristics have been produced for booking work processes in incorporated Grid environment.[10]

Matrix figuring has gotten broadly utilized and is the picked foundation for some logical estimations and undertakings, despite the fact that it requests a precarious expectation to learn and adapt. The registering and capacity assets in the Grid are restricted, heterogeneous and frequently overloaded.[11]

In Grid registering frameworks, the information stockpiling subsystem is actually disseminated among a few hubs and coherently divided between a few clients. This features the need of a) accessibility for approved clients in particular, b) secrecy, and c) honesty of data and information: in one term security. [12]

3. System Architecture and Algorithm

For Distributing Tasks between remote objects and local objects buffering solution is used when Sharing Tasks between The task scheduler object and chunk sender needed. The task scheduler receives tasks from the user code and the chunk sender should distribute them among the Executers. For each task the task scheduler should schedule the tasks for remote executers and ask the chunk sender to send them and wait until this operation is done. The task scheduler object and the chunk sender object each runs a separated thread, the task scheduler write the tasks to task chunks buffer and the chunk sender read and distribute the tasks in the buffer. The same solution used in the Executer Component between chunk receiver object (which receives tasks from the client chunk sender over the network) and task runner object (which Execute the Task). When executer is created, a RMI connection is opened between the System Manager and the executer. When executer is killed, the system removes this connection from it, however, the RMI connection still opened (by itself) for ~10 minutes. So if the user tries to kill the executer, update the system, and then create it again before ~10 minutes, the system will face the same previous problem again (Dynamic loading) since the RMI connection still exists, therefore, the system checks if the System Version (increases every time the user updates the system) and Executer Version (determined by the Task/Result classes) are equal. If not, the system moves the executer to the black list and runs only after the user runs Auto-Update.

The architectural design process is concerned with establishing a basic structural framework for a system. It involves identifying the major components of the system and communications between these

components. In the following sub-sections we delve into the design aspects and the sub systems involved in this software package.

3.1 Context level Diagram

The objective of a system context diagram is to focus attention on external factors and events that should be considered in developing a complete set of system requirements and constrains.

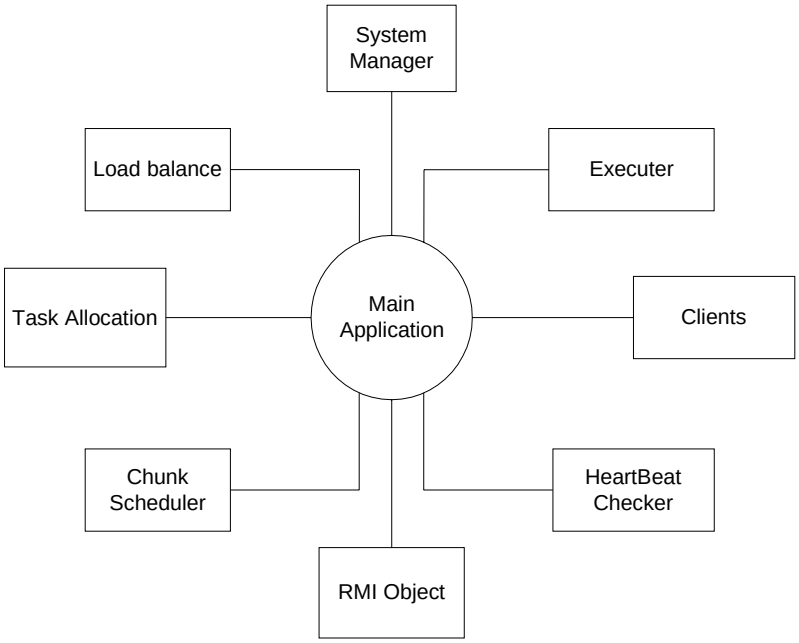

Figure 1: File downloading and distribution using grid computing for application

In the above figure 1, represents the context level diagram for File downloading and distribution using grid computing for application , it contains the modules of system manager, load balance, task allocation, Chunk scheduler, RMI Object, Heartbeat checker, clients, executer.

Main System Manager: holds the main system package including constants, all workers (see Workers System below) and references to all it needs to control such as Executers/Clients Remote Info, Executers List, Black List, Version manager, etc… and his main functionality is match-make between Client Tasks and Executers.

• Client Box: includes all client info needed to be stored in System Manager (Client Remote info among others).

• Executer Box: includes all executer info needed to be stored in System Manager. Executer List: hold a list of the executers in the system.

• Heart Beat checker: responsible for validating connection existence between System Manager and all other links (Executers/Clients) and Manages Black List.

• Version Manager: manages System versions to assure that the system and all its Executers are up-to-date. It also manages versions of jar files (saves it to the disk for reference).

3.2 Class Diagram of System Manager

Figure 2 show the Class Diagram of System Manager,

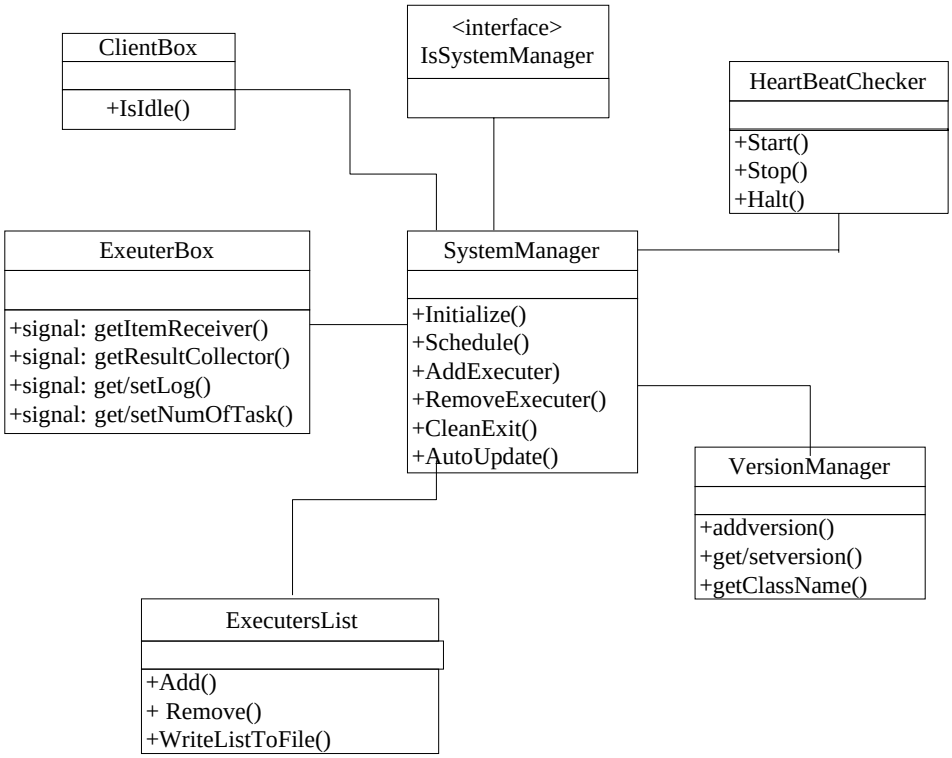

Figure 2: Class Diagram of System Manager

System:

• Command Line Runner: responsible for parsing command line system manager commands.

• File Manager: manages (read, write and create) system files.

• Synchronized Counter: a sync counter used for Clean Exit feature in order to sync between sent Tasks and received Results.

• Logger: log tracer, resided on each Client and Executer. Used basically for debug reasons.

• Remote Info (Client, Executer, RMI): is the basic item holds the needed info to be stored for each Client, Executer and RMI connection (there's a different class for each one).

Networking:

• Network Common: the main network class responsible for RMI connection.
• Security: security connection component.
• RMI base object: bind and unbind RMI connection between two links.
• Item Receiver: receives Items from Clients/Executers and saves them to a concurrent queue.
• Item Collector: collects executed Task Results and returns them to the relevant Client when it ask for results.

System Manager:

As shown in Figure 2 shows the following;
-• Main System Manager: holds the main system package including constants, all workers (see Workers System below) and references to all it needs to control such as Executers/Clients Remote Info, Executers List, Black List, Version manager, etc… and his main functionality is match-make between Client Tasks and Executers.
• Client Box: includes all client info needed to be stored in System Manager (Client Remote info among others).
• Executer Box: includes all executer info needed to be stored in System Manager. Executer List: hold a list of the executers in the system.
• Heart Beat checker: responsible for validating connection existence between System Manager and all other links (Executers/Clients) and Manages Black List.
• Version Manager: manages System versions to assure that the system and all its Executers are up-to-date. It also manages versions of jar files (saves it to the disk for reference).

System Manager UI:

• This component controls the System Manager, manages it by UI. Using it the End-user can see all Executers and Clients (running ones and suspended ones, those who are s in Black List), manage them (Add, Remove and Suspend), see their logs and Auto-Update the system. See System Manager UI in API section.

Client:

As shown in Figure 3 represents the following:

• Client API: implements the API provided to the user.

• Client System: the Client package includes all workers, threads and constants such as Chunk Creator, Scheduler, Collector, Ports and others.

• Chunk Creator: creates a Chunk of Items to send via RMI.

• Chunk Scheduler: responsible for scheduling a Chunk to an Executer

• Result Collector: collects the ready Results for a Client

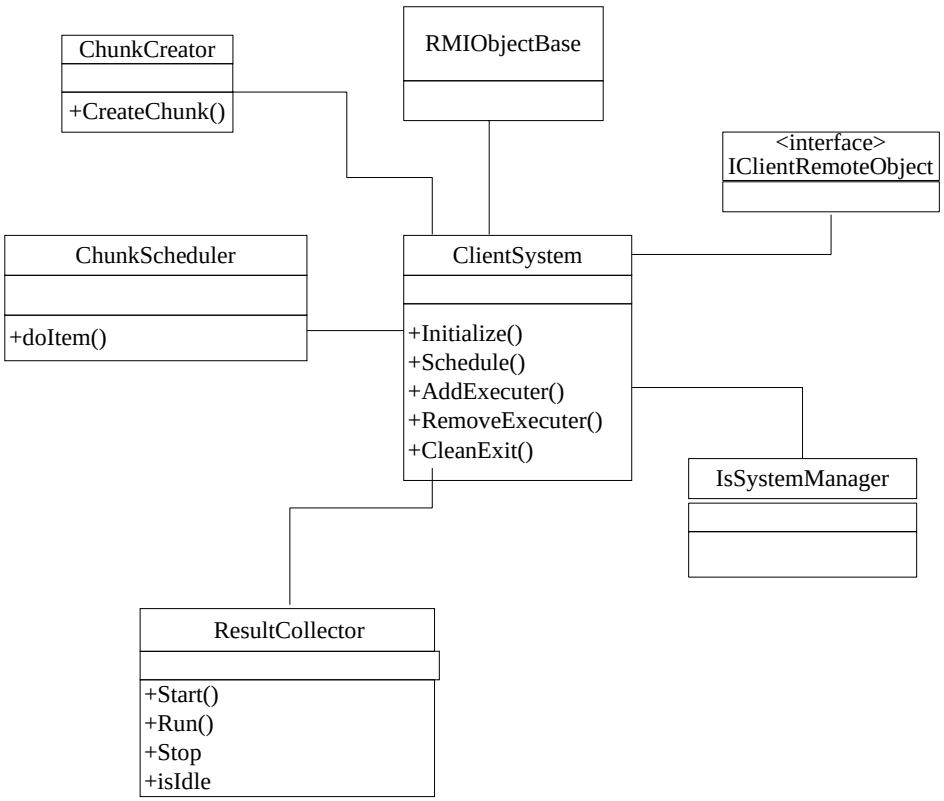

Figure 3: Class Diagram of Client System

Executer:

As shown in Figure 4 :

• Executer System: the Executer package includes all workers, threads and constants such as Directories to save Jar files, Versions, Results Organizer, Task Executer etc…

• Results Organizer: organize the results to Clients (in order to send the Result of each Task to the relevant Client, means the one who added the Task to the system).

• Task Executer: executes the Task after polling it from the Tasks Queue, and puts the Result in the Results Queue.

• Chunk Breaker: breaks Tasks Chunk into single Tasks and puts them in Tasks Queue.

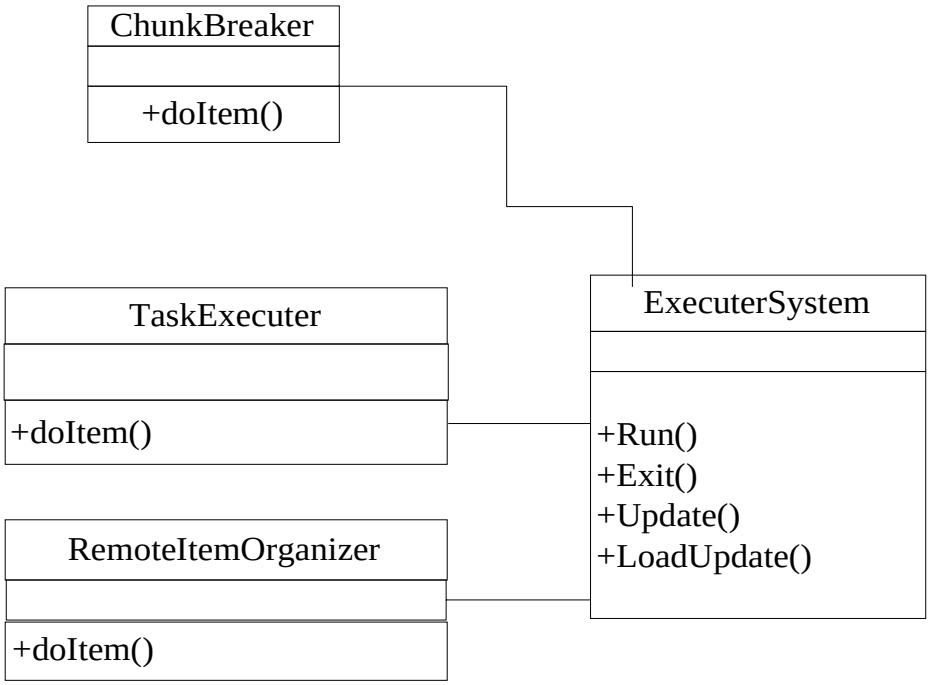

Figure 4: Class Diagram of Executer System

Workers System:

• Factory: creates a worker with a concrete job to do.

• Collection: holds the worker list doing concrete jobs (a Team of workers).

• System: controls a Collection of workers (Starts, Stops and Kills).

• Worker: single worker with its tasks queue and results queue he has to do and already done.

Algorithm for System manager component

System Manager:

1. Initialize a new System Manager Thread running at a particular IP addresses, port, and with a particular heartbeat check interval.

2. START HeartBeatChecker.
3. Create an Executor list file if not already present.
4. Create ClientsMap<ClientInfo , CientBox>
5. Create ExecutersMap<ExecutorInfo , ExecutorBox> //Client and Executor Information
6. Add Executors with IP address and sending and receiving port and maintain a list of active and inactive executors. (HeartbeatChecker used) And create new RMI connection to them.

   [UPDATE TASK]

7. Send update task (AutoUpdateTask consisting of Update jars and New Version to all the Executors.
8. Both Sys Manager and Executors get restarted and new connection is created.
9. Executors get a new Version directory consisting of jar files (from System Manager) needed to perform the required (downloading) task.

   [CLEAN EXIT]

10. Clean exit task is initiated.
11. All the pending tasks are sent to executors and results are returned back to respective clients.
12. A CleanExitTask is sent to all the executors which force them to exit.
13. And SystemManager itself finally exits.

Scheduling performed by SYSTEM MANAGER to distribute Task chunks among the active executors:

Of course the scheduling algorithm has a great impact on the system response time, and this area has got a great scope for improvement as far as this project is concerned, we are as of now using a very simple scheduling algorithm to concentrate on the importance of Load Balancing.

Our implemented algorithm:

1. At first when all the executors are empty provide tasks to all of them in a Round Robin manner.
2. After that provide tasks to the next executor which is least busy i.e. it has least number of pending tasks.
3. Also if an Executor falters i.e. gives some exception while executing any task, the possibility of it being assigned any further task is least.

Future improvement to be done in the algorithm:

1. As system manager keeps all the information about executors including number of tasks sent to each one of them, it is made to hold one more information about executors their Computational Capacity (might be RAM or Processor speed).
2. Scheduling can then be done on the basis of that information i.e. until an Executor gets a maximum number of task, the next task will be sent to it as it has best Computational Capacity.

Executer System:

1. Initialize a new executor system running at a particular IP address and sending and receiving port. Load it with current version and last updated Jar files.
2. Buffers for Task receiving and result sends are created.
3. Create a new WORKERSYSTEM for executor.
4. WorkerSystem consists of a ChunkBreaker worker, ItemReciever worker, Remoteitemorganiser worker, TaskExecutor worker.

   [CHUNK BREAKER]
5. Takes the received chunks from Itemreciever.
6. ChunkBreakerClassifier classifies the tasks in the chunks as normal Download tasks, AutoUpdateTasks, and CleanExit tasks and sends them to proper workers for their interpretation.

   [ITEM RECIEVER]
7. Receives tasks scheduled by SystemManager and stores them in the RecievedTasks Buffer.

   [REMOTE ITEM ORGANISER]
8. Organizes the results obtained after execution in a Hashmap for corresponding clients.

   [UPDATE OPERATION]
9. AutoUpdateTask is received and a new Version file is created.
10. Within that update.jar file which has the class files for download task.
11. It has also the class file name which contains the name of the main executor class which does the main Download operation.
12. Also it contains a jar directory which contains all the Jar files and API's needed for the download task.
13. Finally Executor System is restarted and a new connection with the SystemManager is formed.

14. This provide the system to dynamically adapt to change the task to be executed by just changing the update API's and Update.jar.

    [TASK EXECUTOR]

15. Polls the tasks from the Received tasks Buffer.
16. Does the downloading operation by calling the DownLoadExecutor class. (This resides in the Update.jar received after update operation.
17. Puts the results in Results buffer to be sent back to be organized by RemoteitemOrganiser and sent back to the Clients.

Client System:

1. Initialize a new Client System running at particular IP and port.
2. Start a new worker System consisting of Chunk Creator, Chunk Scheduler, Result Collector.

    [CHUNK CREATOR]

3. Take the tasks entered by the user.
4. Take the Max Chunk Size.
5. Coagulate the tasks into chunks.
6. Also tasks with high priority will be chunked and scheduled first.
7. In this System every fourth is taken to High Priority.
8. Take tasks from TaskInputbuffer and put them in Chunks Buffer.

    [CHUNK SCHEDULER]

9. It is a worker in Client System.
10. Take input from Chunks Buffer.
11. Send to SystemManager.
12. Use the Schedule operation of the SystemManager to assign an executor to every chunk.
13. And finally send the task to the particular executor using assign executor operation of SystemManager.

    [RESULT COLLECTOR]

14. This worker collects the incoming results from every executor.
15. Collected results are first stored in the Results Buffer.
16. From their results are taken and stored in a DocIndexRepository, having a particular root_dir and a downloadList File.

Networking:

1. All the Client, executor and SystemManager systems are created as remote objects, whose services are remotely available through JAVA RMI.
2. All these system instances are loaded in the RMI registry, with their Services being exposed through remote interfaces.
3. All the Systems are continuously running at given RMI port, so that is services can be accessed remotely using other systems.

4. Implementation of the proposed system

Figure 5, shows the connection information, system manager port, current version, status, active executers, inactive executers and clients. It has a functionality to connect or disconnect to the executers and client and has a functionality to show the trace. It also has CleanExit function to exit. shows you System Manager, Executers, Clients and the updated version. It also has an exit button. System Manager manages the chunk size and the chunks for network optimization and getting the result to the right client. The executers are the computers who are actually downloading the file and giving results back to the manager. Clients are the computers who are actually submitting the request in one file. Updates function shows the update version and when a new executer is added the system manager uses this function to make all the clients and all the executers to be in the same version. To get the user interface of the system manager press System Manager button. In SystemManagerGUI click Updates button click browse button and select update.jar file which is inside SystemManager folder and then click Update.

As you click Update Button one frame will appear than click browse button in that frame and select all the jars file inside SystemManager/jars then checked Force Update checkbox then finally click Commit Update and wait for 30 second within 30 second SystemManager and all the running executer will restarted.

Figure 5: Proposed System Manager Control panel

Snippet for Download Files :

```
public class DownloadFiles {

private final String rootDir_;
private final DocIndexRepositorydownloadedDocs_;
private void run(String inputFileName)
throws Exception {
BufferedReader input =
FileHandler.openFileReader(inputFileName);
while(true) {
String line = input.readLine();
if(line == null) break;
String url = line;
if(downloadedDocs_.containsKey(url))
continue;
try{
int docIndex = downloadedDocs_.allocateDocIndex();
String text = downloadAndParseFile(url);
downloadedDocs_.put(url, docIndex);
String fullFileName = rootDir_ + '/' +
DocNameHandler.getFileNamePrefix(docIndex) + ".txt.gz";
```

```
new File(fullFileName).getParentFile().mkdirs();
OutputStreamoutputFileStream =
FileHandler.openFileOutput(fullFileName);
outputFileStream.write(text.getBytes("UTF-8"));
outputFileStream.close();
}
catch(Exception e) {
System.out.println(e.getMessage());
}
}
```

This code creates a directory structure under files/, and store the parsed text in the leaf subdirectories with filenames being serial numbers of the URLs in the list. The list of downloaded URLs and their serial numbers is stored in files/docs.list.txt. This list is checked at the beginning to prevent downloading the same URLs again after the program is stopped and restarted.

It uses the DocIndexRepository API (this is actually used by the DownLoadClient code) for creating a directory structure for storing the downloaded file in Zip format along with proper serial numbers. Encoding used is UTF-8. Other API's used are DocNameHandler which comes under the DocIndexRepository Jar. This handles the name along with proper serial numbers to be provided to the incoming results in order of their arrival.

Snippet for Task and result code:

Task Code:

```
public class DownloadTaskextendsItem {
public String url; //Task's content
public DownloadTask() {
super(0);
this.url="";
}
public DownloadTask(long id, String url) {
super(id);
```

```
this.url=url;
}
public String toString(){
return "Task ID: " + this.getId();
}
}
```

Result Code:

```
import diSys.Common.Item;
@SuppressWarnings("serial")
public class DownloadResultextendsItem {
public String text; //Result's content
public String url; //Result's content
public DownloadResult(long id) {
super(id);
}
}
```

Basically there are three entities in the system which is worked upon Item, Task, Result and Chunk. Above I have shown the code for task and result. Both inherit from the Item base class. Task has a particular URL, Priority and ID. Result has particular ID, URL and Text which contains the actual result.

The Task and result are the actual objects which are used by other classes to submit a task to system manager, schedule it to executors and receive the actual downloaded results. So the submitted tasks are actually DOWNLOADTASK and received results from Executors are actually RESULTS.

Snippet for Download Client:

```
public class DownloadClient {
private static String rootDir_;
private static DocIndexRepositorydownloadedDocs_;
public static void main(String[] args) throws Exception {
//Initialization
RemoteClient<DownloadTask, DownloadResult> client =
```

```
new RemoteClient<DownloadTask,DownloadResult>
("localhost", 5000 /*port*/, 10 /*chunk size*/);
BufferedReader input = openFileReader("bin\\list.txt");
rootDir_ = "Downloads";
downloadedDocs_= new DocIndexRepository(rootDir, false);
client.Start();
//add DownloadTasks
LinkedList<DownloadTask>downloadsList =
new LinkedList<DownloadTask>(); long id = 0;
while(true){
String line = input.readLine();
if(line == null) break else; (String url = line;);
downloadsList.add(new DownloadTask(id++,url));
}
for(DownloadTaskdt:downloadsList){
client.AddTask(dt);
}
//get DownloadResults
for(long i=0; i<id; i++)
{
try {
DownloadResult dr = client.GetResult();
ProcessResult(dr); //Parses the code.
}
catch(Exception e){
e.printStackTrace();
System.out.println(e.getMessage());
}
}
client.Stop(); System.out.println("Client Done!");
}
```

The code is pretty similar to the previous code. client.addTask() method used instead of downloading the web-page itself (and the executer will download the code. Also, client.getResult() method used to get the results) This code is where the execution of the Client system starts. Client takes in the System Manager address and the port number, makes a connection to the System manager thorough Remote Method Invocation creates a DocIndexrepository using the API downloadLibs. This is to create a repository in which all the downloaded files will be stored in a particular order in a particular format. (ZIP format). Then the client system starts all its workers including Chunk creator etc. Then it reads the Tasks from the lists file, provide particular priority to every task. And the Chunk creator creates the chunks of the optimum size, and sends to manager for scheduling. Then finally the Results are taken back and stored in the DocIndexRepository.

Snippet for Executer Code:

```
public class DownloadExecuterimplements
IExecutor<DownloadTask,DownloadResult> {
public DownloadResult run(DownloadTask task) throws Exception {
DownloadTask task=task;
DownloadResult res=new DownloadResult(task.getId());
System.out.println("trying to download url: "+task.url);
//Do the job...
res.text = this.downloadAndParseFile(task.url);
res.url = task.url;
System.out.println("url: " + task.url + " downloaded!);
return res;
}
private String downloadAndParseFile(String url)
throws Exception {
HttpResult result = HttpDownloader.download(url);
if(result.getErrorResponse() != null) {
throw new Exception("Error downloading " + url);
}
String text = sb.getStrings(result);
return text;
```

}

}

The Download Executer simply implements the run (Task t) method called to download a specific URL implements the Iexecutor Interface which is a Remote Interface. Takes a Download Task object, Uses HttpDownLoader API (Downloaded from Sun API repository) to download the File given in the form of a URL , parse the HTML content and saves in text file format. In case of any exceptions the exceptions are thrown and given to System manager for handling. All the Events going on are logged in the system manager logger.

5. Result and Analysis

Result 1:

Chunk Size: 2

Number of executers: 3

Graph below shows response time with changing number of input tasks:

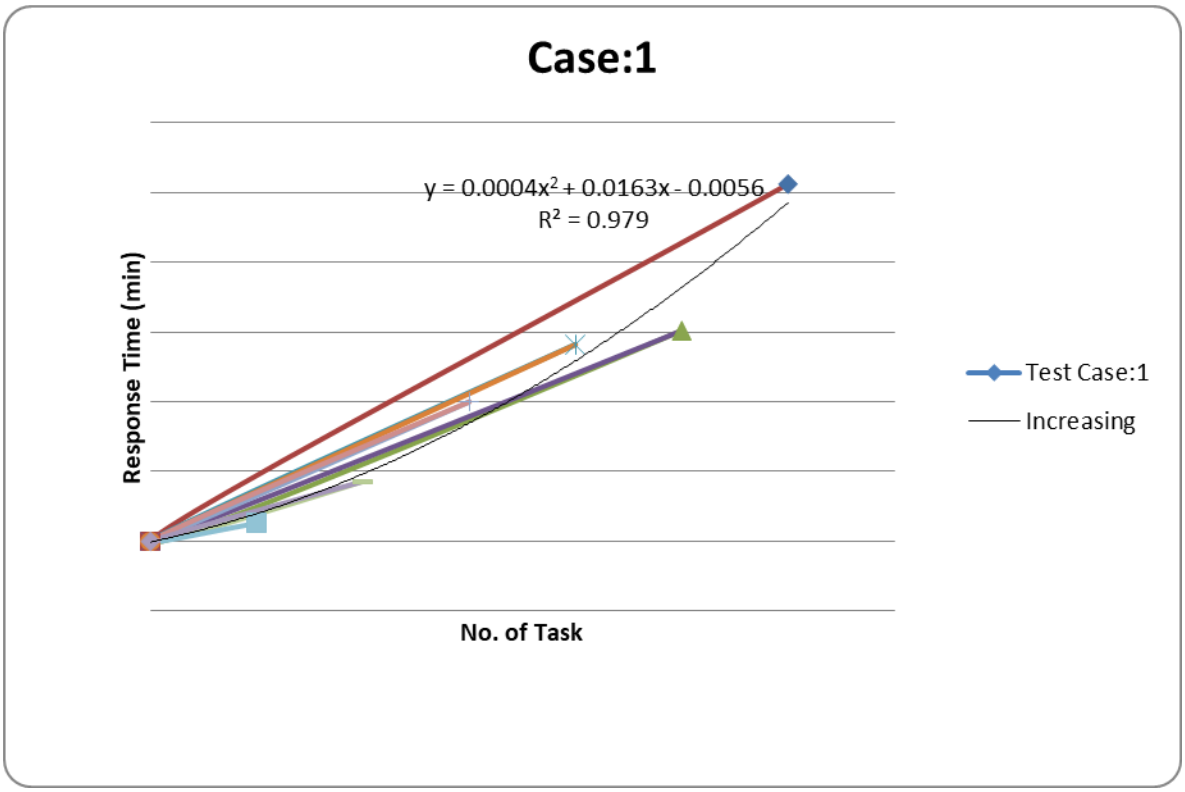

Figure 5: Response Time for variable no. of tasks with constant chunk size

and constant no. of executers

The graphs in Fig 5., shows the test case where the chunk size and number of Executers are constant .These have values 2 and 3 respectively. This graph shows that with increasing no. of task the response time will increase. From the graph we can see that it is the polynomial function having degree 3. Since we are dividing the task according to the task that are already executing the system .the graph becomes polynomial.

Case 2:

Chunk Size: 2

Number of input tasks: 40

Graph below shows response time with changing number of Executers:

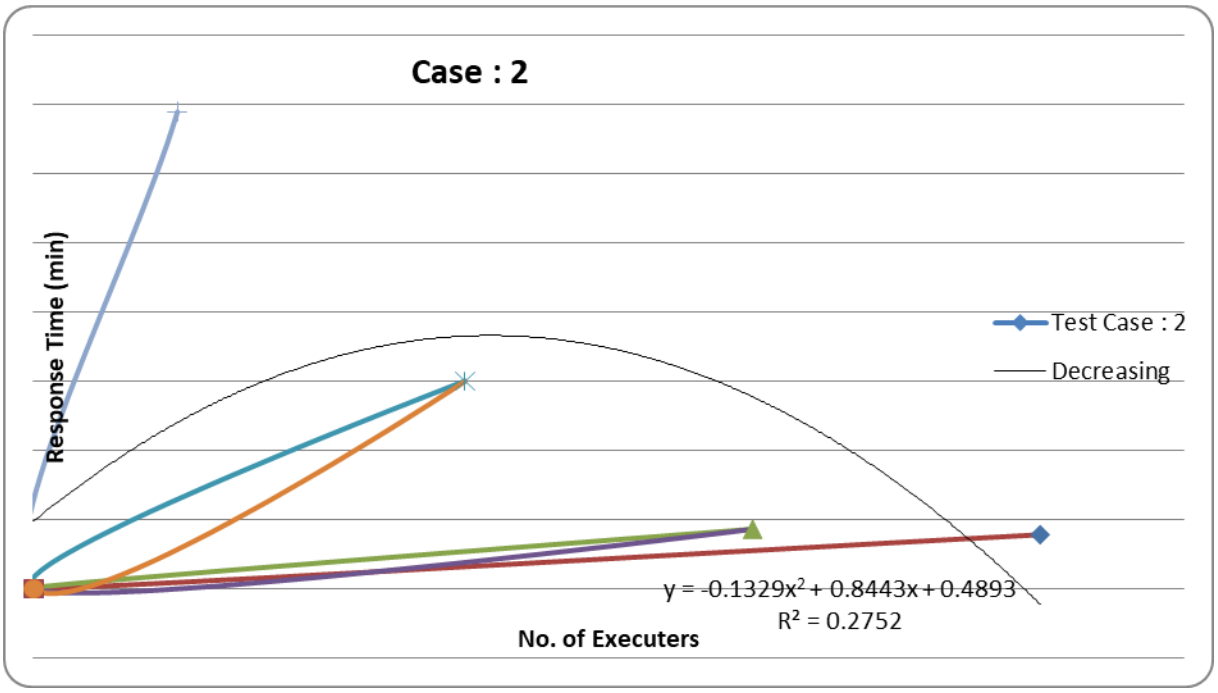

Figure 6: Response Time for variable no.of executers with constant chunk size

and constant no. of tasks

The graph in Fig 6, shows that with chunk size, no. of input task to be constant,(where chunk size =2, Number of input task =40) with increase in the no. of executers the response time will exponentially decrease .In other words if the no. of executers decreases the response time will increase exponentially.

6. Conclusion

Much of the current work in distributed, object-oriented systems is based on the assumption that objects form a single class. This class consists of all entities that can be fully described by the specification of the set of interfaces supported by the object and the semantics of the operations in those interfaces. The class includes objects that share a single address space, objects that are in separate address spaces on the same machine, and objects that are in separate address spaces on different machines (with, perhaps, different architectures). On the view that all objects are essentially the same kind of entity, these differences in relative location are merely an aspect of the implementation of the object. Indeed, the location of an object may change over time, as an object migrates from one machine to another or the implementation of the object changes. The research work is to recognize the implications for software quality of various decisions which must be made in the process of specifying a distributed system. The core logic of the project is to build a generic distributed system which supplies a friendly user API allows remote execution. The prototype is intended for clients who need to execute tasks over more than one computer without getting involved with complex networking APIs. Using the system will allow the client to run his tasks produced over one computer and the system will take care of distributing the tasks over all computers connected to it by their priority in order to be executed in parallel and return the tasks' results to the same computer the developer use deploying Round Robin Algorithm. The system is platform independent; means it can run over every platform since it is written in Java programming language. Moreover, the system can be used in environments where some machines are behind firewalls.

References

1] J. Fu, Z. Shi and Z. Zhang, "An Edge Computing Framework for Digital Grid," 2020 IEEE 3rd International Conference on Electronic Information and Communication Technology (ICEICT), 2020, pp. 670-673, doi: 10.1109/ICEICT51264.2020.9334295.

2] I. COLAK, R. BAYINDIR and S. SAGIROGLU, "The Effects of the Smart Grid System on the National Grids," 2020 8th International Conference on Smart Grid (icSmartGrid), 2020, pp. 122-126, doi: 10.1109/icSmartGrid49881.2020.9144891.

3] R. Guharoy et al., "A theoretical and detail approach on grid computing a review on grid computing applications," 2017 8th Annual Industrial Automation and Electromechanical Engineering Conference (IEMECON), 2017, pp. 142-146, doi: 10.1109/IEMECON.2017.8079578.

4] S. Luo, X. Peng, S. Fan and P. Zhang, "Study on Computing Grid Distributed Middleware and Its Application," 2009 International Forum on Information Technology and Applications, 2009, pp. 441-445, doi: 10.1109/IFITA.2009.249.

5] J. Xie, C. Yang, Q. Huang, Y. Cao and M. Kafatos, "Utilizing Grid Computing to Support Near Real-Time Geospatial Applications," IGARSS 2008 - 2008 IEEE International Geoscience and Remote Sensing Symposium, 2008, pp. II-1290-II-1293, doi: 10.1109/IGARSS.2008.4779239.

6] X. Li, T. Chen, Q. Cheng, S. Ma and J. Ma, "Smart Applications in Edge Computing: Overview on Authentication and Data Security," in IEEE Internet of Things Journal, vol. 8, no. 6, pp. 4063-4080, 15 March15, 2021, doi: 10.1109/JIOT.2020.3019297.

7] A. Aleshinloye, M. A. Manzoor and A. Bais, "Evaluation of Dimensionality Reduction Techniques for Load Profiling Application in Smart Grid Environment," in IEEE Canadian Journal of Electrical and Computer Engineering, vol. 44, no. 1, pp. 41-49, winter 2021, doi: 10.1109/ICJECE.2020.3018433.

8] M. Saini, S. Khan, S. Singh, R. Gupta, P. Upadhyay and S. Soni, "Smart Grid: Problems, Avenues for Study & Attainable Solutions," 2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2021, pp. 513-518, doi: 10.1109/ICACITE51222.2021.9404566.

9] A. I. Kawoosa and D. Prashar, "A Review of Cyber Securities in Smart Grid Technology," 2021 2nd International Conference on Computation, Automation and Knowledge Management (ICCAKM), 2021, pp. 151-156, doi: 10.1109/ICCAKM50778.2021.9357698.

10] M. Rahman, R. Ranjan and R. Buyya, "A distributed heuristic for decentralized workflow scheduling in global Grids," 2009 10th IEEE/ACM International Conference on Grid Computing, 2009, pp. 163-164, doi: 10.1109/GRID.2009.5353050.

11] Krašovec, B., Filipčič, A. Enhancing the Grid with Cloud Computing. J Grid Computing 17, 119–135 (2019). https://doi.org/10.1007/s10723-018-09472-w

12] Cunsolo, V.D., Distefano, S., Puliafito, A. et al. GS3: a Grid Storage System with Security Features. J Grid Computing 8, 391–418 (2010). https://doi.org/10.1007/s10723-010-9157-9